

University of Dundee

## Automatic Recognition of Landmarks on Digital Dental Models

Woodsend, Brénainn; Koufoudaki, Eirini; Mossey, Peter A.; Lin, Ping

*Publication date:*  
2020

*Licence:*  
CC BY-NC-ND

*Document Version*  
Early version, also known as pre-print

[Link to publication in Discovery Research Portal](#)

*Citation for published version (APA):*  
Woodsend, B., Koufoudaki, E., Mossey, P. A., & Lin, P. (2020). *Automatic Recognition of Landmarks on Digital Dental Models*. <https://arxiv.org/abs/2012.12946>

### General rights

Copyright and moral rights for the publications made accessible in Discovery Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from Discovery Research Portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain.
- You may freely distribute the URL identifying the publication in the public portal.

### Take down policy

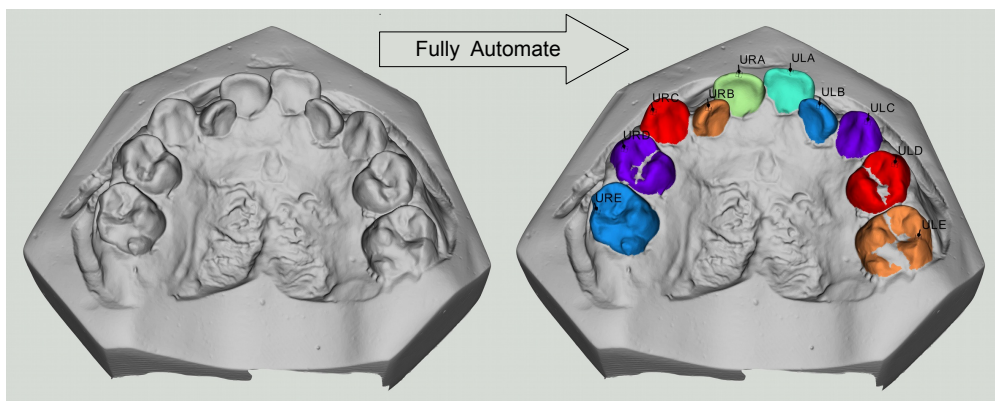
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# Graphical Abstract

## Automatic Recognition of Landmarks on Digital Dental Models

Brénainn Woodsend, Eirini Koufoudaki, Peter A. Mossey, Ping Lin

arXiv:2012.12946v1 [math.NA] 23 Dec 2020



# Automatic Recognition of Landmarks on Digital Dental Models

Brénainn Woodsend<sup>a</sup>, Eirini Koufoudaki<sup>b</sup>, Peter A. Mossey<sup>b</sup>, Ping Lin<sup>a,\*</sup>

<sup>a</sup>*School of Science and Engineering, University of Dundee, Nethergate, Dundee DD1 4HN, United Kingdom*

<sup>b</sup>*School of Dentistry, University of Dundee, Nethergate, Dundee DD1 4HN, United Kingdom*

---

## Abstract

Fundamental to improving Dental and Orthodontic treatments is the ability to quantitatively assess and cross-compare their outcomes. Such assessments require calculating distances and angles from 3D coordinates of dental landmarks. The costly and repetitive task of hand-labelling dental models impedes studies requiring large sample size to penetrate statistical noise.

We have developed techniques and software implementing these techniques to map out automatically, 3D dental scans. This process is divided into consecutive steps – determining a model's orientation, separating and identifying the individual tooth and finding landmarks on each tooth – described in this paper. Examples to demonstrate techniques and the software and discussions on remaining issues are provided as well. The software is originally designed to automate Modified Huddard Bodemham (MHB) landmarking for assessing cleft lip/palate patients. Currently only MHB landmarks are supported, but is extendable to any predetermined landmarks.

This software, coupled with intra-oral scanning innovation, should supersede the arduous and error prone *plaster model and calipers* approach to Dental research and provide a stepping-stone towards automation of routine clinical assessments such as "index of orthodontic treatment need" (IOTN).

**Keywords:** Dental, Landmarks, 3D Analysis, Automation, Artificial Intelligence

---

## 1. Introduction

Three dimensional digital imaging has emerged as a new tool in clinical practice, and provides opportunities for improving research in multiple directions. In dentistry specifically, 3D analysis of jaws and dentition for treatment planning and treatment outcome assessment has already been gold standard (significantly before the digital era) in daily practice, especially in disciplines like orthodontics.

That has been achieved through dental impressions followed by construction of three-dimensional plaster dental models. These models are then analysed by identifying on them landmarks and measurements of specific parameters. That manual process has always been problematic, as it is time consuming and is subject to both random and systematic errors. Recently intra-oral scanners have been developed that can deliver high accuracy digital models of single teeth and full dental arches. These digital models don't require storage, can be shared without needing to be shipped, and by annotating landmarks, lend themselves to digital analysis. [2, 12, 14]. In this paper, we explore techniques to automate the finding of these landmark features on a digital 3D model of

---

\* Corresponding author

Email addresses: bwoodsend@gmail.com (Brénainn Woodsend), e.koufoudaki@dundee.ac.uk (Eirini Koufoudaki), p.a.mossey@dundee.ac.uk (Peter A. Mossey), p.lin@dundee.ac.uk (Ping Lin)

sets of teeth and the landmark 3D coordinates.

Accurate automated dental landmark identification would be a great tool both to researchers of dental science, and in routine treatment planning and assessment in clinical dentistry. Tooth measurements have always been regarded as time-consuming [6], and so automatic landmarking would save time for dentists, and opens up the possibility of studies on large numbers of teeth sets.



Figure 1: Manual measurements on plaster models

## 2. Background

Traditionally dental impressions are a negative imprint taken from the patients mouth using an impression tray and a setting paste. By many patients this is considered a very uncomfortable procedure. Then the negative imprints are poured with plaster, which then sets creating 3D plaster copies of dental arches. These arch copies are then used to measure and estimate tooth position in relation to neighbouring and opposing teeth (Figure 1). The development of digital impressions have been a game changer, as it can develop dental arch imprints of high accuracy very quickly and with greater comfort for the patient. The digital data can either be collected directly from the patient or

by creating a plaster model first that can then be scanned. Of course the first option is much more efficient in multiple ways, including time, materials, storage management and human resources.

The 3D scans produced by the intra-oral scanners are in STL format – a standard open source format for 3D models. They describe only the surface of a model and therefore require wholly different analysis techniques to voxel based **volumetric** scans such as the output of a micro-CT or 2D images such as X-ray.

The original objective of this study is to automate the modified Huddart and Bodenham system (MHB) [11] for assessing treatment of cleft lip/palate. This system has been shown to be far more objective and reproducible than its predecessors [3], but has had limited uptake by clinicians due to the considerably extended time it takes. These traits make it a prime target to be converted into an automated software. Ma et al [10] devised a semi-automatic system, automatically calculating an MHB score based on manually selected coordinates of key landmarks. These particular landmarks are the midpoints of the incisors, the tips of the canines and the outer cusps (bumps) on the molars. Manual landmark placing is time consuming, requires expertise and is prone to human error. We have created a software which is set to identify dental landmarks in accordance with the MHB scoring system (although the software can be adjusted to work with many systems). The aim of the application is to increase efficiency and automation of the scoring of dental surgical outcomes, encouraging a more efficient workforce in global dental care. By moving from traditional plaster “hard-copy” models to 3D digital models, the global burden of care will be reduced. In addition, the reliability and repeatability of dental model scoring will improve by reducing human error and increasing the accuracy of measurements.



### 2.1. Similar prior arts

There have been several studies that have worked towards similar or overlapping objectives as those of this work. Some of the most interesting and partially relatable ones to our development are summarised here.

One of the oldest systems, developed by Kumar et al [8], aims for fully autonomous mapping out of a dentition in a slightly different methodology than the one followed by us. An orientation step is not part of the procedure as it is assumed that all models are oriented in the same way. Then they introduce a *watershed* method to partition the teeth. This method is analogous of flooding a mesh with water until small lakes are formed – these lakes are the teeth – with the catch that the *height* of a mesh vertex is in-fact defined based on mixture of surrounding curvature as well as its regular geometric height. Teeth are then identified using curvature of cross sections. The watershed method has persisted, being reused in many more modern works.

Considerable work has been put into tooth segmentation, often with the *fully automatic* constraint relaxed to *semi-automatic*. This is largely driven by forensic scientists who wish to identify postmortems where dentitions were only partially recovered. Kronfeld et al [7] have designed a *snake* algorithm to *walk* along the edges of teeth to partition them, with some safeguards to help jump gaps in tooth outlines. Zou et al [4] have developed a tooth partitioning system based on user-supplied tooth labels. It turns the mesh into a graph network (dubbed *a harmonic field*) with each vertex a node and each edge an arc. Each unlabeled node has an unknown potential and each labelled node has a fixed potential dependent on its label. Each arc has a weight derived from curvature, and flow through it proportional to the difference in potentials of its two vertices and inversely proportional to its weight. The whole system of vertex potentials/arc

flows is solved with the constraint: the net flow of each non-labelled vertex must be zero. A vertex is part of a tooth if its potential exceeds some halfway threshold. Each tooth must be separated one at a time, but if you solve the linear system using sparse LU factorisation, the potentials of each labelled vertex can be altered with negligible extra computation i.e. the processor-intensive part is done only once. This method truly tackles the issue of *shabby* casts where the tooth edges are poorly defined or have gaps.

Lastly, Kalogerakis et al [5] focuses on simultaneous segmentation and labelling of arbitrary objects (not dentally related) using machine learning. They manage it with great success using training sets of less than 10 models per object type.

## 3. Materials/Methods

The software was developed in the programming language Python. We had available 239 dental models of different types (listed in table 1). The models are stored in STL format.

An STL file describes the only surface of a 3D object, making it hollow. It contain no colour or texture information. Once read from file, it is typically referred to as a *mesh* - an unordered list of triangles with each triangle defined by the  $(X, Y, Z)$  values of its three vertices. Since all triangles are represented separately, vertices that are corners of multiple triangles are duplicated. A typical first step on reading an STL, which our software adopts, is to find and enumerate all the unique vertices to make the connections between neighbouring triangles easier to find. Traditionally, STL files contain no scale information and their units are generally arbitrary, but in dental scans the units are always millimetres.

The data analysis for the successful automated landmark identification, is a multi step process approach. The steps undertaken by the software are summarised by the flow chart in figure 2.



Count	Qualifiers			
24	Upper			
21	Lower			
17	Lower	IO		
16	Upper	IO		
81	Upper	Primary	Cleft	
80	Lower	Primary	Cleft	

Table 1: Types and counts of models used in this study

### 3.1. Find Orientation

Before any kind of analysis can be done, the model's orientation needs to be established. As different scanners use wholly different orientation conventions and initial the orientations aren't accurate to begin with, it is important that the system is able to find and internally normalise the orientation of models before they can be further analysed.

Orientation is expressed as a series of *unit-vectors* representing *left*, *right*, *backwards*, *forwards*, *down*, *up* and *occlusal*. These unit-vectors should be derived from the model and any inspection of a vertex should use these vectors so that the effect of the model's position is negated. i.e. To get the height of a vertex take the inner product of that vertex with the *up* vector instead of examining raw Z values (or Y values on a TRIOS-scanned model). The model itself must not move or the relative positioning between a patient's upper and lower jaws will be lost.

Section 3.2.1 requires that the vertical directions are particularly accurate. The horizontal directions are less crucial. To generate the required unit-vectors a PCA (principle component analysis – see section 3.1.1) based method was designed.

#### 3.1.1. Principal Component Analysis

PCA looks at covariance (spread from the centre of mass<sup>1</sup>) in all directions. It returns a set of axes (three perpendicular unit-vectors) ordered from most

<sup>1</sup>The centre of mass being the middle of the model, or more precisely, the mean of all of its vertices.

to least covariance. PCA may be used to find the directions of the longest, middle-length, and shortest dimensions of an object. PCA calculations are available in the appendix Appendix A.

A dental model is wider (left/right) than it is long (forwards/backwards) and longer than it is tall. So the output of PCA on a dental model should yield *left/right* as the first (largest covariance) unit-vector, *forwards/backwards* as the second and *up/down* as the third.

PCA's advantages are: (i) it is one of most popular statistical techniques and thus the function of implementing it is directly available in any common programming environment; (ii) the model can be positioned and orientated anywhere and PCA will track it indifferently.

#### 3.1.2. Signs of the unit-vectors

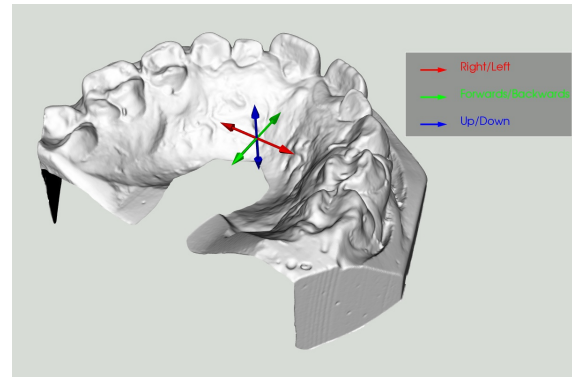


Figure 3: The unit-vectors from PCA. The red arrows represent *left* and *right*, but PCA doesn't specify which is *left* and which is *right*. Likewise with *forwards/backwards* (green) and *up/down* (blue).

The sign of an eigenvector is arbitrary and consequently so are those of the unit-vectors found using the above (see figure 3). The signs have to be checked by other means and the vectors reversed if they are wrong. The following checks were adopted (in the following order).

**1. Vertical *up/down/occlusal*** As the occlusal surface is the most detailed, the density of triangles (and their corresponding unit-normals) there is highest. A

mean of all the model's unit normals should therefore approximately point *occlusally* and PCA's *occlusal* can be sign matched to this approximate *occlusal*.

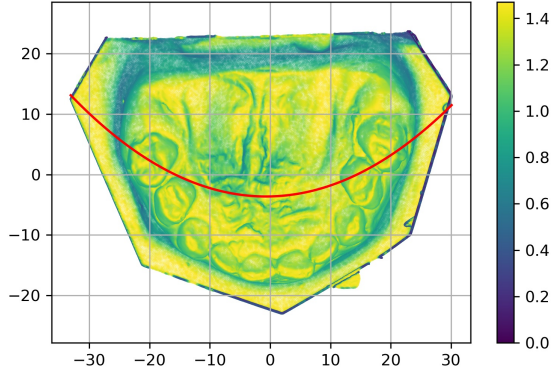


Figure 4: Quadratic weighted fit (red line) used to test the sign of the *forwards* unit vector. The line's being U shaped tells us that *forwards* is actually *backwards* and needs reversing.

**2. Forwards/Backwards** The software extracts the horizontal components of the mesh's vertices and approximately captures the *arch* shape of the jaw. All the points on the mesh are reverse weighted by their unit normals' agreement with *occlusal*. This is supposed to emphasise the labial and lingual surfaces. A weighted quadratic curve is fitted to the horizontal components. If the *forwards* vector's sign is correct then the quadratic should be  $\cap$  shaped with a negative  $x^2$  coefficient. If it is  $\cup$  shaped then the *forwards* vector needs reversing.

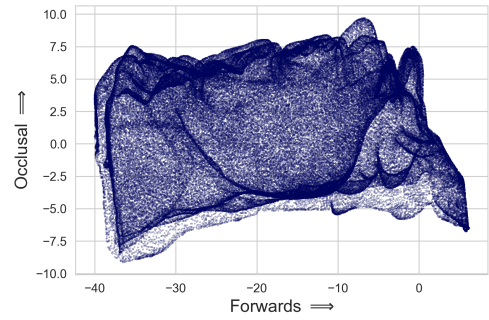
**3. Left/Right** With the signs of the other two axes known this can just be determined so as not to mirror the model. Treating the unit-vectors as column vectors, mirroring can be checked for using:

$$\det \begin{bmatrix} \hat{\mathbf{e}}_{\text{right}} & \hat{\mathbf{e}}_{\text{forwards}} & \hat{\mathbf{e}}_{\text{up}} \end{bmatrix} = \begin{cases} 1, & \text{Non-mirroring} \\ -1 & \text{Mirroring} \end{cases} \quad (1)$$

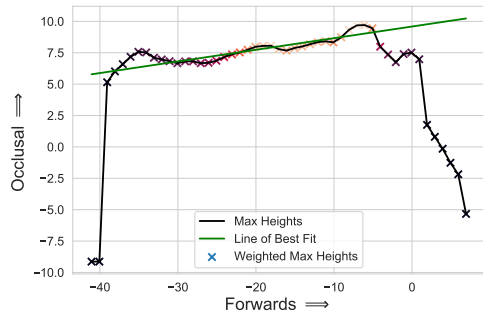
If it mirrors then reverse  $\hat{\mathbf{e}}_{\text{right}}$ .

### 3.1.3. Fine-Tuning the Vertical Axis

Later steps (mostly section 3.2) require particularly accurate vertical unit-vectors. Depending on the type of dental model (intra-oral scans and plaster models with rough bases), the vectors from PCA are typically inadequate. Using the PCA *forwards* and *occlusal* unit-vectors (after the above sign checking), more accurate unit-vectors can be found by fitting a straight line through the top-most outline (see figures 5a and 5b). To do this, divide the mesh horizontally into bins and find the highest point in each bin. Then fit to those highest points, aggressively weighting the centre-most (horizontally) and highest points so as not to be affected by dips due to missing teeth, or the drop-off at the front and back of the model.



(a) Side-on flattened screenshot of all the points on a model. This is a lower jaw with the patient looking to the right.



(b) The top-most outline (black line) of the same side on view, the weighted points along that outline ('x' shaped markers) and the line fitted to those points (green line) which will represent horizontal.

Figure 5: Method to fine-tune the vertical axis

### 3.2. Find the Peak Points

Most of the landmarks required for MHB scoring are either located on teeth tips or cusps. Mathematically these can be described as local maxima in the occlusal direction or any vertex that is higher than all its immediate neighbours. These points are referred throughout this article as *peak points* or just *peaks* (see figure 6). No information about what feature each peak point represents is found at this stage. There will be many peak points that do not represent an actual landmark and must be cleared away in later steps. The only requirement of this step is to land at least one point on each tooth or cusp (for molars and premolars).

#### 3.2.1. Filtering Based on Height

To avoid finding large numbers of peaks on the gums, base (for plaster models) and roof of the mouth, the search area is reduced to only the top (occlusally) of the model so as to mostly include only teeth. Any points 6mm or more below the highest tip of the teeth are excluded from the search area. The height threshold is visualised with the transparent blue planes in figure 6.

The height threshold chosen (6mm below the highest peak) is comfortably low enough to include the tips or cusps of all teeth of interest – even if they are chipped (although these may be mistakenly rejected in section 3.5 – the tooth assignment stage). Setting it very low increases the number of non-tooth features picked up. These features will be safely removed later (in section 3.3.4) but at considerable computational expense.

### 3.3. Partition into Individual Teeth

#### 3.3.1. The General Idea

This step uses *curvature* to find the boundaries of teeth. Curvature is a quantitative measurement of how much a surface deviates from being flat at a particular point. The exact definition of curvature can

vary – the one chosen is signed so that an outside corner (a bump, cap or tip) is positive and an inside corner (a slot, groove or crease) is negative. The join between each tooth and the gum is a crease and therefore the curvature along the join is negative (see figure 7).

Starting at the top of a tooth and recursively including adjacent mesh triangles until an edge of significantly negative curvature is hit, one can find all triangles that are part of that tooth. This region covered will be referred to as the starting point's *region*. Each peak point found in the last section is used as a starting point. Peaks on the same tooth will have regions that overlap. By testing for overlapping regions, duplicity of teeth is avoided. Any peaks that weren't on a tooth to begin with will not be bounded by the creases of tooth-gum joins and will therefore try to include most of the model if left unchecked. By imposing the rule *stop if travelled more than a tooth's width away from the starting point*, and testing if that rule was actually used, non-tooth peaks and their corresponding regions can be identified. These non-tooth regions are labelled *spilled* and are discarded.

Thus, each tooth should come out nicely partitioned without any duplicity. (No information about which tooth it is which is found here.) And all non-tooth features should remove themselves.

**Complications** Whilst the above may seem promising it doesn't happen in practice. Below is listed some of the more prominent issues which must be solved.

- Non-tooth features often don't spill. This happens mostly on the palatal rugae and on *knobbly* plaster models.
- Each cusp of the molars and premolars will usually be separated and have to be re-grouped back together.

The flowchart in figure 8 outlines the full series of

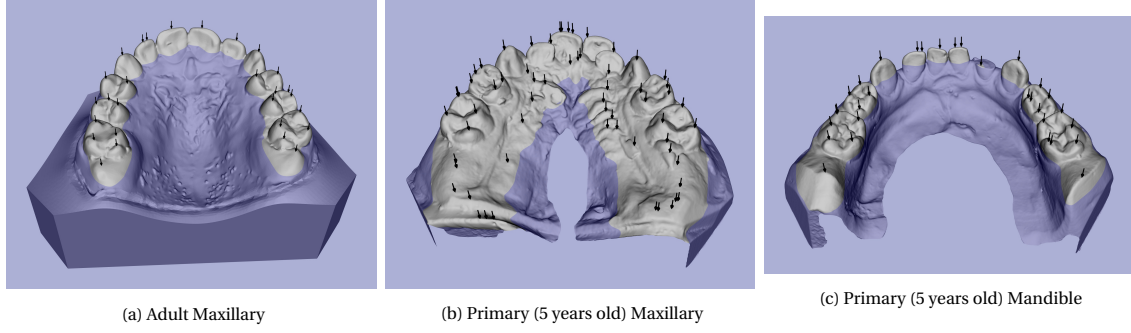


Figure 6: The peak points (black arrows) and the height threshold (blue tint) found for three different models.

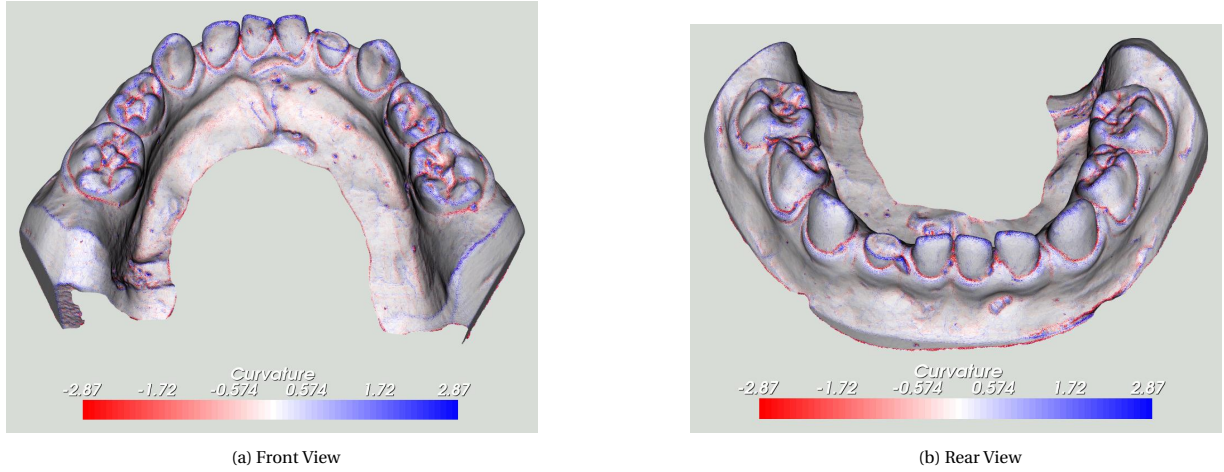


Figure 7: Signed Curvature – Notice the ring of red along the tooth-gum join surrounding each tooth and between neighbouring teeth.

steps undertaken.

### 3.3.2. Flood Filling From Each Peak-Point

**Calculating Curvature:** Curvature comes in many different forms. For continuously differentiable surfaces, it can be calculated using high derivatives of the surface geometry. But for a discretely defined surface, like that of an STL file, approximating high derivatives gives a very poor signal to noise ratio. Prior works have typically used Principal Curvatures which are defined per-vertex. Principal Curvatures are derived by either taking minimum [9, 15] or mean [13, 7] of nearby edges' curvatures.

Our system uses per-edge curvature rather than per-vertex curvature. Edge curvature was chosen as it is easier to calculate, involves slightly less

approximating and it makes the after-analyses easier as each triangle always has exactly three edges and three neighbours (triangles in non-closed meshes will occasionally have less neighbours) whereas, with per-vertex curvature, each vertex can have any number of neighbours and edges, requiring awkward *ragged-array* data structures.

Our form defines the curvature ( $\mathbf{k}$ ) at an edge by comparing the outward unit normals ( $\hat{\mathbf{n}}_0$  and  $\hat{\mathbf{n}}_1$ ) of the two triangles on either side of that edge along with the distance  $|\Delta\mathbf{x}|$  between the centres of each triangle, ( $\Delta\mathbf{x}$  being the displacement from the centre of triangle 0 to the centre of triangle 1.)

$$\mathbf{k} = \frac{\hat{\mathbf{n}}_0 \times \hat{\mathbf{n}}_1}{|\Delta\mathbf{x}|}$$



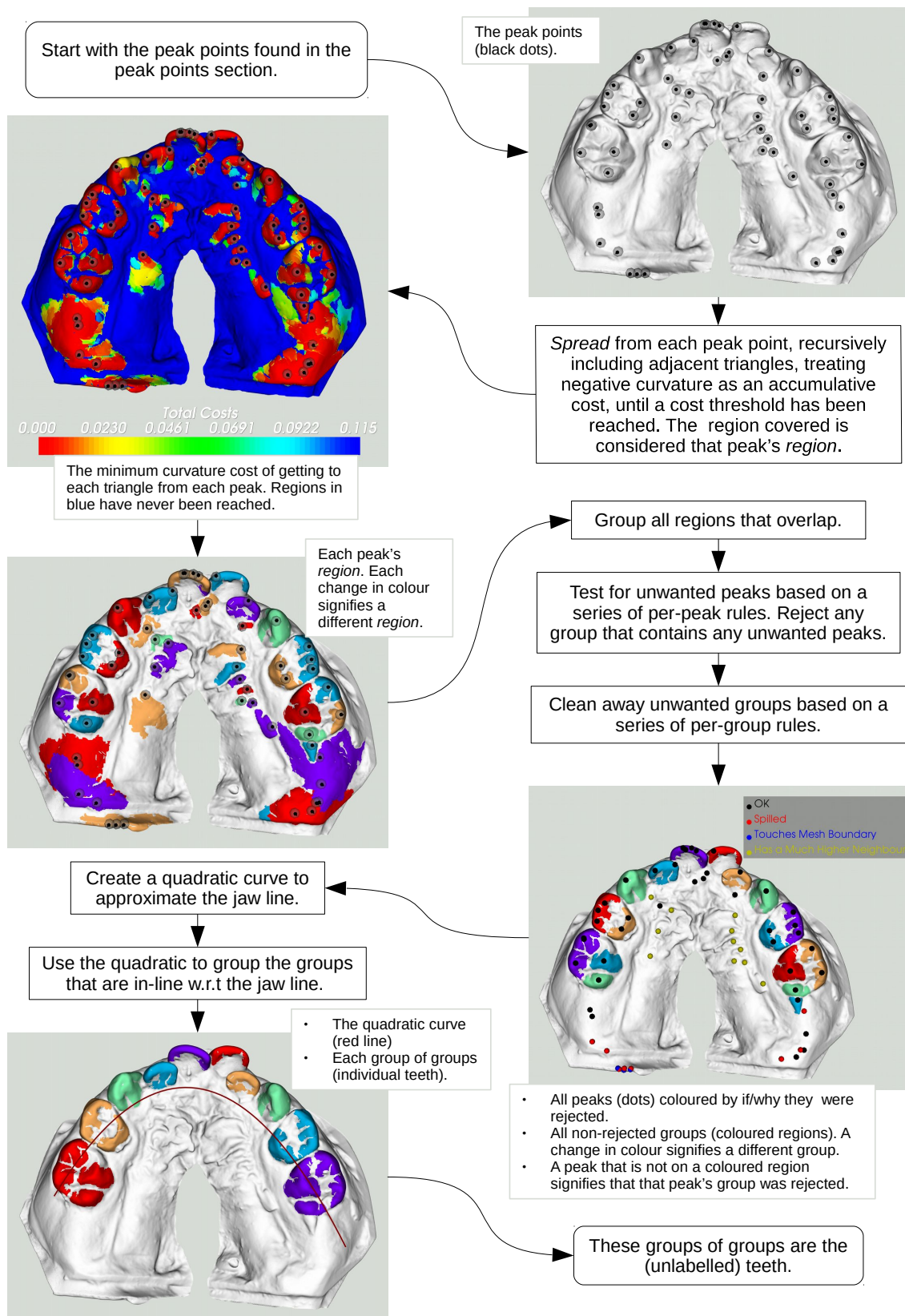


Figure 8: Flowchart summary of the automatic landmark finding process

This equation yields a vector, the direction of which is just the edge's direction (which we don't need). The magnitude quantifies how tight the corner of two mesh triangles sharing the edge is. This still leaves no information about the sign. The equation used to get the sign is displayed below:

$$\text{signed curvature} = -\text{sign}(\hat{\mathbf{n}}_0 \cdot \Delta \mathbf{x})|\mathbf{k}|$$

The output of the above equation is a signed curvature scalar, for each mesh triangle, for each of its adjacent triangles (as shown in figure 7).

A **Cost Map** is derived from the above signed curvature. Only creases (negative values) are relevant. And, being a *cost*, all values should be positive.

$$\text{cost} = \max(-\text{signed curvature}, 0)$$

Again, this cost map is per edge of each of the mesh's triangles. The cost of including a new adjacent triangle to the region is the cost of crossing the triangle edge separating that triangle from the region. Costs accumulate making a problem analogous to *The Shortest Path Problem* from Graph Theory [1] with the exception that we are interested in the cheapest path to every mesh triangle rather than a single destination. In this analogy a mesh triangle is considered as a graph node and the weight or distance on the edge that connects two nodes (or triangles) is the cost defined above. Once the cumulative cost (i.e. the minimal cost solved from the shortest path problem) to every triangle is found it is compared to a *cost threshold*. Any triangles with cumulative costs below the threshold are included as part of that peak's *region*. The *cost threshold* is solved for dynamically per model so as to maximize the total surface area classed as part of a tooth after the grouping and cleaning stages throughout the rest of this section.

Mathematically, the problem is formulated as follows. Let:

- $T[i]$  be the total cost of reaching triangle  $i$  (with  $i \in [1, \text{number of triangles in the whole model}]$ ).
- $E[i, j] \geq 0$  be the curvature based edge cost of crossing the edge from triangle  $i$  to triangle  $j$ , a neighbour of  $i$ .
- $T_{max} > 0$  be our curvature threshold.

Then solve the following for all elements of the vector  $T$ .

- $T[i] = 0$  if any of triangle  $i$ 's vertices is the initial peak point.
- $T[i] = \min_j(T[j] + E[i, j])$  if  $\min_j(T[j] + E[i, j]) < T_{max}$ .
- $T[i] = T_{max}$  otherwise.

A triangle  $i$  is then considered part of the region if  $T[i] < T_{max}$ .

Rather than truly solving the system as a linear algebra problem, which would be difficult due to the uses of min, and slow due to the large number of triangles involved, a far more efficient algorithm was devised. This algorithm is very close to Dijkstra's algorithm [1] for solving the Shortest Path problem.

1. First, initialise all  $T[i]$ s to  $T_{max}$ .
2. Initialise an empty queue.
3. For each triangle  $i$  which contains the starting peak, set its  $T[i]$  value to 0 and add its three neighbouring triangles to the queue.
4. Pop (choose and remove) an element  $i$  from the queue.
5. Evaluate  $t = \min_j(T[j] + E[i, j])$  for that  $i$  and its three neighbours  $j$ . It doesn't matter that one or more of the  $T[j]$  may not have been processed yet. If  $t < T[i]$  then set  $T[i] = t$  and add the three neighbours  $j$  to the queue to be (re)calculated later.



6. If the queue is empty, terminate. Otherwise return to step 4.

### 3.3.3. Group Overlapping Regions

Group all regions that overlap (i.e. have triangles in common). Indirect grouping is allowed, meaning that if regions *A* and *B* overlap, and regions *B* and *C* overlap, but *A* and *C* don't, then *A*, *B* and *C* should still form one group.

### 3.3.4. Remove Non-Tooth Regions

Non-tooth regions are tested for with a series of rules to remove unwanted features. It's ideal, but not imperative, that all non-tooth features are removed before proceeding with the tooth assignment section.

#### Per Peak or Per Peak's region Rules:

A peak and the group it belongs to is rejected if it meets any of the following criteria:

- It has *spilled* (as defined in section 3.3.1).
- It is in close horizontal proximity to a much higher peak. Or more precisely, if the ratio of the vertical displacement  $\delta V$  and horizontal distance  $\delta H$  to any other peak is greater than 1.5 then this peak is almost certainly gum.

Whilst these rules could've been applied earlier, by waiting until after overlap-grouping, unwanted peaks which are harder to filter are often grouped with obviously non-tooth peaks and can therefore be removed safely.

#### Per Group of Regions Rules:

A group of overlapping regions is removed if any of the following apply:

- Group contains only lingual or buccal pointing surface normals. Any tooth should have both a lingual and a buccal side, or for very slanted teeth, at least a significant variance. The groups

on the rugae will all face only palatally so will be rejected by this rule.

- Group touches the mesh boundary. This is primarily for intra-oral scans which often pick up bits of cheek which must be ignored.

### 3.3.5. Group Inline Groups

This is the second of the two grouping stages.

Labial/buccal and lingual/palatal cusps of molars and premolars will often still be separated but can be put back together by grouping by position along the jaw-line (again allowing indirect grouping). The output groups of groups should be whole teeth.

The arch of the jaw-line makes the geometry of the above deceptively awkward. A quadratic curve, fitted to the horizontal components of the remaining peak points, approximates the jaw-line (see section 3.4 for more information on the quadratic). Each region group can be mapped onto the quadratic to determine its span (left-most and right-most position) along the jaw, effectively 1-dimensionalising (flattening) the jaw line. These spans can be compared directly to test if two groups are inline.

### 3.3.6. The Output and its Drawbacks

The resulting groups of groups from above are teeth and are shown in figure 9.

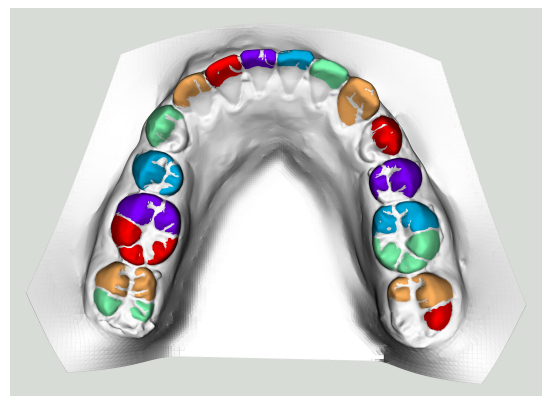


Figure 9: The final output of this tooth segmentation method. Each change in colour signifies a different tooth.

The only teeth that may still remain split are the mesial and distal halves of the molars (see figure 9). These half-molars will need to be merged but we can't do that yet until they are identified as such. Otherwise we may mistakenly glue two premolars together instead. This correction is handled by the tooth assignment step (section 3.5).

Lingual portions of incisors, and the middles and grooves of molars are all inside corners and therefore are unrecognised as parts of the teeth. An adapted version of the method described by Bei-ji Zou et al [4] has been applied as an additional step to capture to whole teeth with good success. But as this has little impact on finding MHB landmarks, and depends on some heavy-weight sparse linear algebra libraries, applying this step has not been adopted as the default behaviour.

### 3.4. Best-fit Quadratic Orientation Curve

In sections 3.3 and 3.5 it is important to be able to perform operations that refer to the arch shape of the model. Namely, to define the directions tangential (mesial and distal) and normal (lingual and buccal) to the jaw's arch, and to facilitate sorting by position around the arch.

To do these requires a continuously differentiable curve fit of the arch. Other works have used cubic-splines to do this [8, 10]. However, it assumes that the points fitted to are ordered and contain no outliers/anomalies (neither assumptions hold here). The jaw is roughly quadratic shaped – so a simple quadratic fit based on the least squares method was chosen.

#### 3.4.1. Construction

The curve is fitted to the horizontal components of the peak points from section 3.2 with  $x$ -increasing defined as left to right across the mouth and  $y$ -increasing as going forwards.

$$y = ax^2 + bx + c \quad (2)$$

Most of the time, the fit is tolerant enough that the peak points do not require any filtering beforehand - but not always. It is therefore best to wait until after the peak cleaning from section 3.3.4 has removed those irrelevant points before applying this technique.

#### 3.4.2. Ordering Peaks and/or Teeth

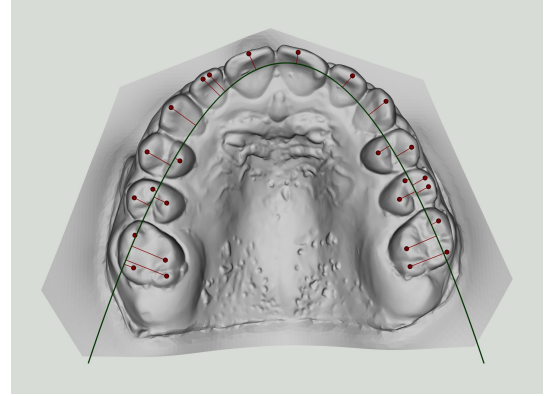


Figure 10: The least-squares quadratic curve (green line) fitted to the peak points (red dots). Each peak is projected back onto the curve (red lines). The peaks can be ordered by where their projections lie on the curve.

Any series of points, such as the peak points or the centre of mass of each unlabelled tooth, can be objectively sorted and enumerated using the quadratic. To do this, project each point to its nearest point on the curve, then sort and enumerate by the  $x$  value of each nearest point (see Figure 10).

#### 3.4.3. Generating Distal and Buccal Unit-Vectors

The directions *along the jaw-line* and *perpendicular to the jaw-line* can be defined with reference to the curve (see figures 11a and 11b). The following commonly required direction vectors can be derived.

- *Mesial* (towards the front teeth) tangent to quadratic with positive  $y$  component.
- *Distal* (towards the back teeth) tangent to quadratic with negative  $y$  component.

- *Buccal/labial* (outwards towards the lips) normal to quadratic with positive y component.
- *Lingual/palatal* (inwards towards the tongue) normal to quadratic with negative y component.

The y axis is guaranteed by section 3.1.2 to point forwards, so a negative  $\frac{dy}{dx}$  indicates that the gradient and tangent of the quadratic curve is in the direction of the back of the jaw. As an example, *distal* is calculated in full below.

$$\text{distal tangent} = \begin{cases} -\left[1, \frac{dy}{dx}\right] & \text{if } \frac{dy}{dx} \geq 0 \\ \left[1, \frac{dy}{dx}\right] & \text{if } \frac{dy}{dx} < 0 \end{cases}$$

To convert the direction vector back to 3D use the *right* and *forwards* unit-vectors from section 3.1 and the x and y components of the vector as follows:

$$\text{3D distal} = x\hat{\mathbf{e}}_{\text{right}} + y\hat{\mathbf{e}}_{\text{forwards}}$$

At any point in space, the nearest point to the quadratic can be solved for, the tangent or normal be calculated at that nearest point and a unit-vector representing distal (see figure 11a), buccal (see figure 11b), lingual or mesial can be generated.

### 3.5. Tooth Assignment

Section 3.3 yields a collection of unlabelled sub-samples of the original mesh which will be referred to throughout this chapter as *blobs*. Each blob is either a whole tooth, one half of a molar, or occasionally a non-tooth feature. The blobs are sorted and enumerated (with direction left to right) by section 3.4.2. This section assigns a tooth type to each blob if appropriate. The aimed results are shown by figure 12.

Complications that may arise and need dealt with by the method:

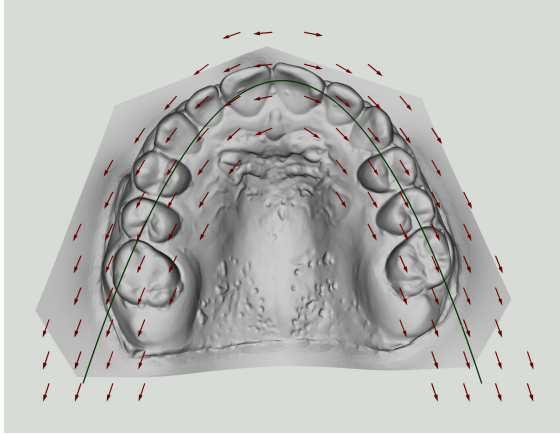
1. Teeth may be missing, and may or may not leave a space where the tooth would have been.
2. Molars may be whole, or split into mesial and distal halves, or partially erupted so that only the mesial half is visible.
3. On adult teeth, the 7s and 8s are much less likely to be present.
4. There will be occasional non-tooth blobs.
5. And of course, all the inconsistencies of individual teeth:
  - Natural genetic deviation.
  - Worn, chipped, malformed or re-crowned tips.
  - Teeth tilted at extreme angles or rotated within their socket.

#### 3.5.1. Prelude – Methods That Didn't Work

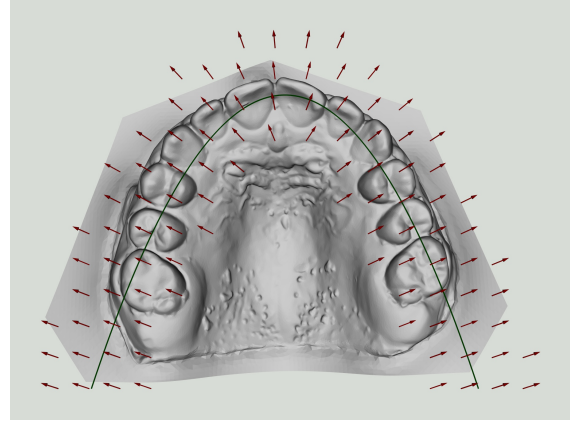
Troubleshooting took many unsuccessful attempts until it was possible. A brief history of those attempts is described to demonstrate how the solution was reached.

The first tempting solution is to enumerate round from the centre. This does not work as any missing tooth will lead to wrong count. Furthermore, the centre of the arch isn't precisely known. Absolute position around a jaw line is too inaccurate to be used - an error of a few degrees can lead to picking the wrong tooth.

Another apparent solution may be to try and use specific tooth characteristics (e.g. number of cusps, geometric properties e.t.c.) to classify teeth. Rule based logic doesn't handle well large varieties of exceptions (or it becomes too complex). The complexity is considerably increased by the range of orientations of teeth. These orientations could be factored out if they were known, but to calculate a tooth's orientation requires that which tooth it is can be known in advance - a *chicken and egg* case.

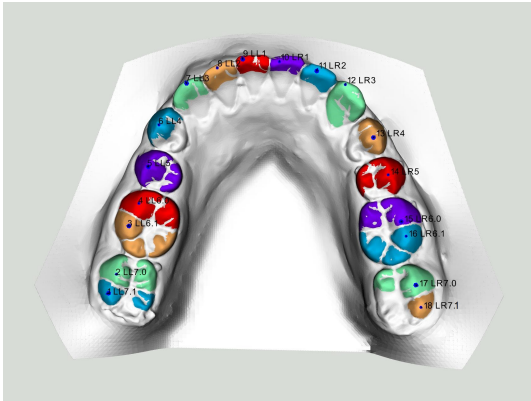


(a) The arrows indicate the *distal* direction generated by finding tangents to the curve.



(b) The arrows indicate the *buccal/labial* direction generated by finding outward normals to the curve.

Figure 11: The *distal* and *buccal* directions can be defined at any point in space using the least squares quadratic curve.



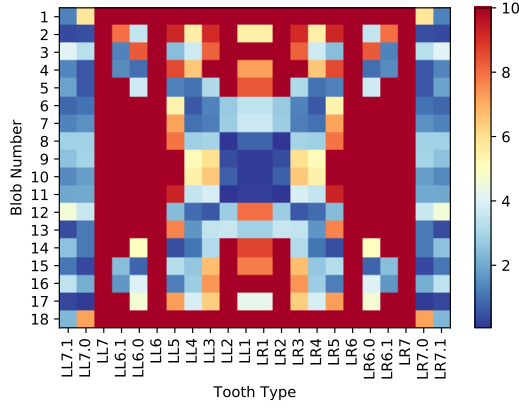


Figure 13: Heat map of assignment costs. Each square represents how unlikely a particular blob (unlabelled tooth-like sub-sample of the mesh) is to be a particular tooth type. Values higher than 10 are clipped to 10 for visual clarity – costs can be much higher.

never definitively say that blob  $x$  is or isn't a given type – only how unlikely it is.

**Solve the Cost Table:** The table of costs is *solved* to find the *optimal*, defined as lowest possible total mismatch, valid assignment. Thus making the final decision as to which tooth is which. The optimal assignment is found using *Linear Programming*.<sup>2</sup>

**The tooth types:** searched for are referenced using Palmer notation, e.g. UR2 for upper right 2<sup>nd</sup> incisor, with an extension to describe half-molars: UR6.0 and UR6.1 represent the mesial and distal halves of an UR6. A list of potential tooth types is generated based on the model's dentition type (permanent/deciduous and upper/lower). The tooth types for each jaw type are listed in table 2.

<sup>2</sup>Linear Programming is a wide-spread field of computational mathematics used to minimize (or maximize) a linear objective function subject to linear equality or inequality constraints.

Jaw Types							
Adult				Deciduous			
Upper	Adult	Lower		Upper	Deciduous	Lower	
UL1	UR1	LL1	LR1	ULA	URA	LIA	LRA
UL2	UR2	LL2	LR2	ULB	URB	LIB	LRB
UL3	UR3	LL3	LR3	ULC	URC	LIC	LRC
UL4	UR4	LL4	LR4	ULD	URD	LID	LRD
UL5	UR5	LL5	LR5	ULE	URE	LLE	LRE
UL6	UR6	LL6	LR6	ULE.0	URE.0	LLE.0	LRE.0
UL6.0	UR6.0	LL6.0	LR6.0	ULE.1	URE.1	LLE.1	LRE.1
UL6.1	UR6.1	LL6.1	LR6.1				
UL7	UR7	LL7	LR7				
UL7.0	UR7.0	LL7.0	LR7.0				
UL7.1	UR7.1	LL7.1	LR7.1				
UL8	UR8	LL8	LR8				
UL8.0	UR8.0	LL8.0	LR8.0				
UL8.1	UR8.1	LL8.1	LR8.1				

Table 2: All tooth types searched for, for each jaw type.

Some clarifications to make here:

- Equivalent teeth from different jaw types such as {UR1, LR1, URA, LRA} are treated independently as if they weren't related.
- Tooth sub-types (incisors, canines, premolars, molars) are similarly ignored. An UR1 is (to the software's mind) unrelated to an UR2.
- Both halves of a molar as well as the whole molar are thought of as independent whole teeth throughout most of this method. The *solve the cost table* stage adds additional constraints to the linear programming model stating that a whole molar and its two halves are mutually exclusive.
- Whilst creating the cost table, left tooth types are considered equivalent to their corresponding right types (making the cost table symmetrical), but whilst solving it, they are searched for separately.

### 3.5.3. Create a Cost Table

An unknown-blob tooth-type pair is tested by comparing the blob to a database of hand labelled blobs (training set). Rather than try to compare whole blob meshes directly (which was tried unsuccessfully), measurements of the meshes are taken and compared. These measurements are referred to in this paper as *tooth characteristics*. They must be applicable to any tooth type and yield a single number per blob.

## Tooth Characteristics

The simplest example is the Area tooth characteristic which measures the total surface area in  $\text{mm}^2$  of a blob. The total surface area has been measured on all the hand-labelled blobs in the training set and the results were grouped by tooth type (see figure 14). These values are referred to as *reference values*. When analysing an unknown blob, the surface area is calculated for that blob (the *test value*) and compared to each group of reference values. The more the test value differs from the reference values, the higher the cost for that tooth type.

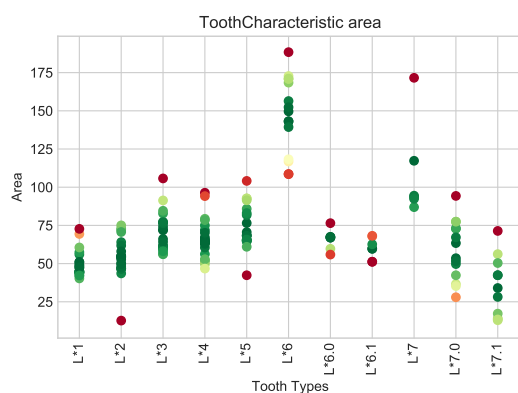


Figure 14: The surface area of each tooth in the training set, grouped by tooth-type. The (\*) placeholder in the tooth-type names symbolises that left and right teeth can be used together.

To demonstrate, suppose an unknown blob had a surface area of  $75\text{mm}^2$ .

- 75 is an ideal value for the blob to be a 3, 4 or 5, or the mesial half of a 7. Costs should be small, typically  $< 1.0$ .
- 75 is on the high side for the blob to be either of the incisors or the distal halves of the molars. Costs should be medium, typically  $2.5 - 5.0$ .
- 75 is far too low to be either whole molar. Costs should be very high, typically  $10.0 - 100.0$ .

This is done for each unknown blob, for each tooth characteristic. The costs for each characteristic (listed

in table 3.5.3) are averaged to give an overall cost for each blob tooth-type pair.

This design makes it possible to provide overall, qualitative trends about tooth types without turning them into strict rules which must apply to every tooth. For example, one might say *canines and premolars have pointed tips but incisors have flat tops*. On its own this statement is unhelpful to a computer because it's not always true. A worn canine is not pointy and would confuse a *pointiness* rule into thinking it's an incisor. But turning it into tooth characteristic overcomes the potential complication. First, a method to quantify a blob's pointiness must be devised. The tooth characteristic will then handle everything else. It will:

- Train itself by applying the method to all labelled blobs in the training set to understand exactly how pointy each tooth type should be.
- Handle the comparison of test values with reference values.
- Self evaluate its reliability. A tooth may be worn, so pointiness is not wholly indicative. Through use of the error metric (section 3.5.3), less trustworthy characteristics are given less weight in the final outcome.

There is no limit to how many characteristics can be used. Generally, the more the better, but it transpired that not many are necessary. The list chosen is given in table 3.

Name	Description	Goal
Total Surface Area	Calculates the total surface area of the blob. This is very easy to implement reliably.	
Mesiodistal Width	As the name suggests. This requires a <i>distal</i> unit-vector which can be generated as described in section 3.4.3.	
Buccolingual Width	Similarly, this requires a <i>buccal</i> unit-vector	This separates premolars and molars from incisors and canines far more reliably than counting cusps.
Pointiness	Calculated by measuring the mesiodistal width at 1mm from the top and comparing it to the mesiodistal width from above.	Helps to distinguish canines from incisors.

Table 3: The tooth characteristics currently used.

A few other characteristics that didn't work are noted in table 4 to deter anyone from trying them again. The results were too noisy to be of use and in some cases were very processor demanding.

Name	Description	Goal
Symmetry	Mirrors the blob them aligns and compares the mirrored version to the original using Iterative Closet Point Algorithm.	Premolars are very symmetric mesiodistally whereas half-molars aren't.
Total Curvature	Add all the unsigned curvatures together.	Supposed to rather lazily separate based on how textured each tooth is.

Table 4: Some old, less successful tooth characteristics.

### Cost Metric – Comparison Function

The cost metric quantifies the mismatch between a test value and a set of reference values (for a single characteristic). The main reason for this choice of metric was to avoid any arbitrary weights that have to be machine-learned.

The metric is based on the mean square error (MSE), which is a common default in machine learning. For a test value  $t$  and vector  $\mathbf{r}$  of length  $k$  of reference values:

$$MSE(t, \mathbf{r}) := \frac{1}{k} \sum_{i=0}^k (t - r_i)^2$$

In order to be able to meaningfully combine

and compare costs and to better solve this optimal assignment problem using Linear Programming, a couple of modifications are required to conform to the following requirements. For a given set of reference values:

1. The minimum possible cost should be zero. With MSE, it depends on the reference values.
2. The output costs should be scale independent. With MSE, a characteristic with large typical values will dominate another with smaller ones.
3. High inter tooth-type variance in the reference values should reduce the costs for test values, thus making the characteristic more lenient.

Requirement 1 is trivial to achieve. Just find the minimum possible cost  $C_{min}$  and subtract it from future calculated costs. Both 2 and 3 can be achieved simultaneously by feeding the metric each of the reference values as test values, taking mean  $\overline{C_{ref}}$  of the resulting costs and dividing through any future costs by this mean. The final formula is written below:

$$C(t) = \frac{MSE(t, \mathbf{r}) - C_{min}}{\overline{C_{ref}}}$$

#### 3.5.4. Solve the Cost Table

The lowest costing possible assignment that is *valid* is found using Linear Programming. General purpose linear programming solver packages are freely available. This implementation uses PuLP. All that is required is to formulate the problem in the mathematical format that is standard in linear programmings. The results are shown in figure 15. LP problems can easier to build from scratch than to read but the gory details of the LP problem are included below.

#### Decision Variables

Assignments are represented with Boolean (true or false) variables. Let  $D$  be an  $m \times n$  table of Booleans,



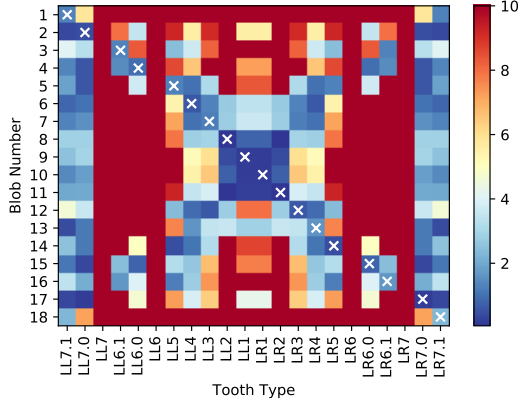


Figure 15: The heatmap from figure 13 with the chosen *optimal* assignment (white X markers). The X markers always form a downward diagonal to ensure the tooth-types are in order along the jaw and will aim to be only on dark blue (low costing) squares. This is from the same model as figure 12 so the X markers here match the annotations there.

where  $m$  is the number of blobs and  $n$  is the number of tooth types.

$$D[i, j] = \begin{cases} 1, & \text{If the } i^{th} \text{ blob is of the } j^{th} \text{ tooth type.} \\ 0, & \text{Otherwise.} \end{cases}$$

A blob may not be a tooth. And a tooth type may be missing. These need to be accounted for.

$$NT[i] = \begin{cases} 1, & \text{If the } i^{th} \text{ blob is non-tooth.} \\ 0, & \text{Otherwise.} \end{cases}$$

$$MI[j] = \begin{cases} 1, & \text{If the } j^{th} \text{ tooth-type is missing.} \\ 0, & \text{Otherwise.} \end{cases}$$

### Constraints

All the rules of dentistry must be expressed as mathematical constraints.

Each blob can either be exactly one tooth type, or it could be non-tooth. So for each  $i = \{1 \dots m\}$ :

$$\sum_{j=1}^n D[i, j] + NT[i] = 1$$

Each tooth type can either appear exactly once or that tooth type is missing: For each  $j = \{1 \dots n\}$ :

$$\sum_{i=1}^m D[i, j] + MI[j] = 1$$

Tooth types should appear in the correct order. To express this mathematically we utilize the fact that both the tooth types and the blobs are ordered from left to right along the jaw. To enforce the order we then need only to ensure that the assignment table  $D$  does not reorder them. If blob  $i$  is of tooth type  $j$ , then no previous blobs can be of further right tooth types and no later blobs of further left tooth types. So if  $D[i, j]$  then none of,

$$D[i', j'], \quad i' = \{1, \dots, i-1\}, \quad j' = \{j+1, \dots, n\}$$

$$D[i', j'], \quad i' = \{i+1, \dots, m\}, \quad j' = \{1, \dots, j-1\}$$

This mutual exclusiveness can be expressed mathematically by adding the variables and applying an upper bound ( $\leq$ ) to the sum.

$$D[i, j] + \frac{1}{m \times n} \sum_{i'=1}^{i-1} \sum_{j'=j+1}^n D[i', j'] \leq 1$$

$$D[i, j] + \frac{1}{m \times n} \sum_{i'=i+1}^m \sum_{j'=1}^{j-1} D[i', j'] \leq 1$$

The  $\frac{1}{m \times n}$  is necessary so that multiple elements inside the double  $\Sigma$  sum can be true simultaneously without violating this constraint.

A whole molar must be missing if either of it's half molar types are not missing: For each molar,

$$MI[\text{whole}] + \frac{1}{2}(MI[\text{mesial}] + MI[\text{distal}]) \geq 1$$



Note the use of  $\geq$  instead of  $=$ . This is because if a patient was missing the molar then the LHS of the equation above would be 2. Similarly, if the molar were partially erupted then the LHS would be 1.5.

### Objective Function

The objective is to minimise the total mismatch cost which is the sum of each entry in the cost table whose corresponding entry in the assignment table is true (left half of equation 3 below).

There also has to be some form of penalty for marking blobs as non-tooth and/or marking tooth-types as missing. Without such a penalty, the optimal solution, with a net cost of 0, would unconditionally be to mark all the blobs as no-tooth and all teeth as missing. Penalising for missing tooth-types gives the advantage of being able to vary the penalty for different tooth-types (right half of equation 3). This provides a very convenient way to tell Linear Programming that 7s and 8s (wisdom teeth) are significantly less common than the other tooth types.  $P[j]$  from above represents the probability that a patient will have a  $j^{th}$  tooth-type, which can be trivially derived from the training data.

Minimise:

$$\sum_{i=1}^m \sum_{j=1}^n D[i, j] \times C[i, j] + 8.0 \sum_{j=1}^n P[j] \times MI[j] \quad (3)$$

The penalty multiplier (or *fussiness factor*) 8.0 is arbitrary. It controls how atypical a tooth can be before it is assumed to be non-tooth.

## 4. Results

This section serves as a graphical results section and highlights some of the more prominent issues either tackled or still to resolve. Throughout this section, all models are coloured by the output from the tooth assignment step (i.e. one colour per tooth, no colour for an unrecognised tooth) and marked with an annotated black cross-hairs on each MHB

landmark.

### 4.1. Orientation

Orientation is easily the most reliable step despite its having the least information to work with. There was only one model, shown in figure 16, for which it didn't work. In this case the cause was a double-cleft which is so heavily textured (giving it a higher vertex density than the teeth) that it dominates PCA.

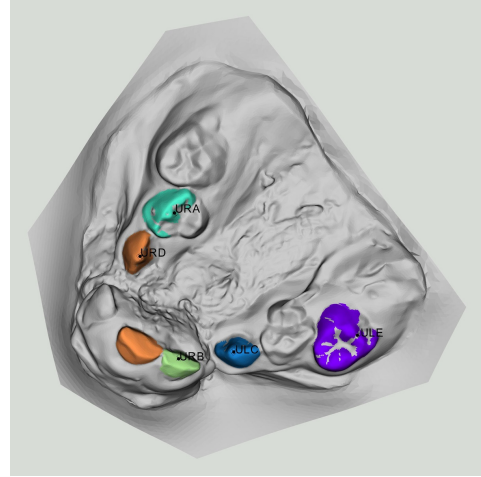


Figure 16: The only model that failed at the orientation step. This photo is oriented with what the software mistakenly thought was the front of the model at the top of the image. Of course, with the orientation wrong, every subsequent output is nonsense.

### 4.2. Tooth Partitioning

Tooth partitioning (section 3.3) gave mixed degrees of tolerance to unclear boundaries and rough tooth surfaces. The adaptive curvature threshold allows it to use a low threshold for models with weak outlines or a high threshold for models with noisy/bumpy tooth surfaces. However, it can not do both simultaneously so a model with at least one poor outline and one bumpy tooth surface will always lose at least one of them. Flattened incisor tips (very common) form dimples at the top giving the same affect as bumpy teeth i.e. it forces the the curvature threshold up so that the algorithm can cross the dimples to reach the

rest of the tooth. Figure 17 demonstrates exactly this case. This is the most common error that the software makes.

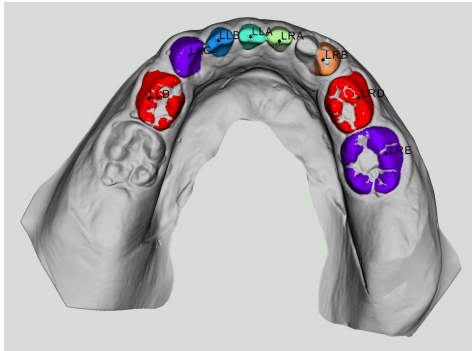


Figure 17: Model with both flattened incisors and a weak outline around the LLE resulting in the LLE not being recognised.

Perhaps a more resilient algorithm in the future will be able to choose thresholds per-tooth rather than per-model. Our attempts to do so generally compromised the cleaning steps in section 3.3.4.

The software exceeded expectations on some really poor quality casts (see figure 18).

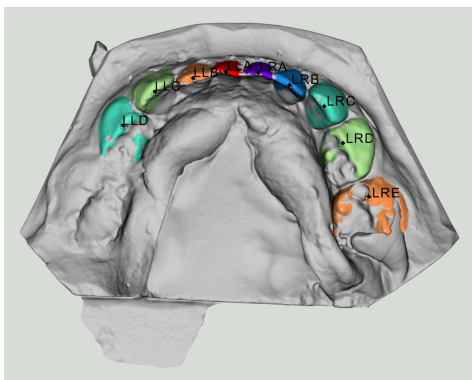


Figure 18: A very poor impression of an infant.

Its tolerance is of course finite. See figure 19). As a side-note: Perhaps, before any further work is done on automating their interpretation, some investigation into getting better impressions/scans from infants would be appropriate.

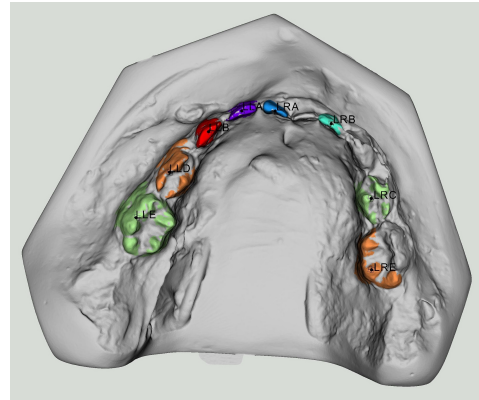


Figure 19: Our very worst plaster model. Whilst the software has found more teeth than we might expect, it is very difficult to find landmarks on teeth that are as poorly recognised as the LRE.

#### 4.2.1. Crowding

It took some coaxing of overlap thresholds in section 3.3.5 but we were able to get good performance for tooth crowding. The difficulty is that lingual and buccal halves of premolars and molars are paired up only because they are in the same position around the jawline so two incisors that are sufficiently crowded together will be mistakenly grouped. We've managed to give it enough tolerance to allow cases like the one shown in figure 20 but, by design, this algorithm will always fail for cases such the one in figure 21.

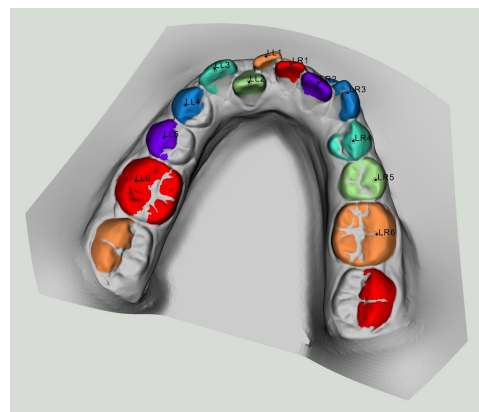


Figure 20: A model with some crowding. The software is tolerant to this degree of crowding (but not much more).

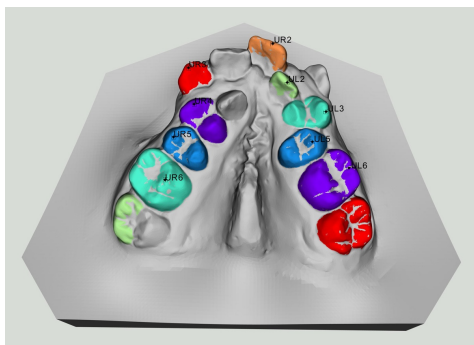


Figure 21: A model with too much crowding for the software. The position of UL2 behind the UL1 directly contradicts the assumption, made in section 3.3.5, that such a positioning implies that those two teeth must in-fact be two parts of the same tooth e.g two cusps of a pre-molar. After tooth partitioning mistakenly decides that the UR1 and UR2 are one tooth, tooth assignment then, on failing to find a tooth type that resemble this strange *double incisor* tooth, rejects it as non-tooth.

#### 4.2.2. Intra-oral scans

Intra-oral scans impose quite a different set of problems. The tooth outlines are, provided they are scanned properly, much crisper but the tooth surfaces often obtain a *fuzzy* texture which negates the advantage of clear outlines. It is primarily for these models that the adaptive curvature threshold was needed. The chosen threshold is typically much much higher for intra-oral scans.

Another problem is the *trimming* or where the scan stops. Partially scanned bits of cheek, lips or tongue often appear in these scans and collect peak points which need to be ignored. A simple *reject anything that touches the mesh boundary* rule (section 3.3.4) easily gets rid of them as in figure 22. However, there is a downside to this – by definition any *tooth* which touches the mesh boundary is lost. Thus, with this rule in effect, clinicians are required to scan to the base any teeth they wish to analyse. To see the effect of this compare figure 24 to figure 23.

In our dataset of intra-oral scans, it was rare that any incisors, canines or premolars weren't fully scanned but about 15% of molars were lost because of this. To fully capture a molar requires getting

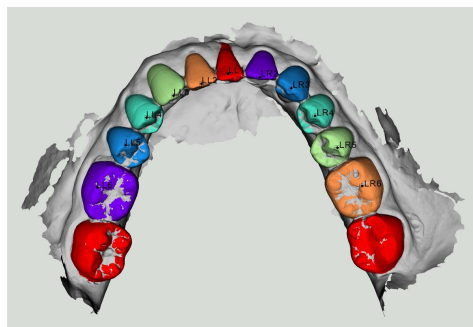


Figure 22: An intra-oral scan with somewhat chaotic trimming. The software is able to correctly ignore it.

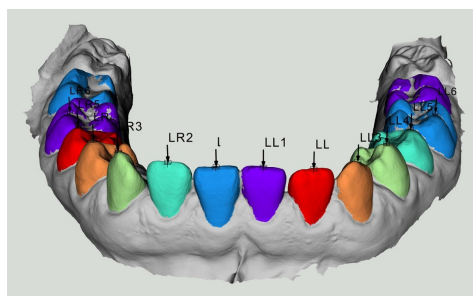


Figure 23: An nice intra-oral model. The scan includes a non-zero amount of gum surrounding each tooth. Partitioning works OK.

the scanner behind it to capture its distal side – an uncomfortable procedure. This also has the potential to waste a lot of clinician's time should a patient need to be rescanned because of a small gap in a tooth capture.

#### 4.3. Tooth assignment

The 1<sup>st</sup> half of the assignment step (section 3.5.3) is quite weak. For models with most, if not all, teeth present the linear programming (section 3.5.4) picks up the slack to give a good end result but for models with many teeth missing, such as the one shown in figure 25, the software gets progressively less reliable.

The assignment works much better for deciduous models (albeit partly because it has less teeth to decide on) despite there being considerably more variation in deciduous teeth. Lower incisors are an exception (see figure 26) but, given that even experienced clinicians struggle to distinguish lateral

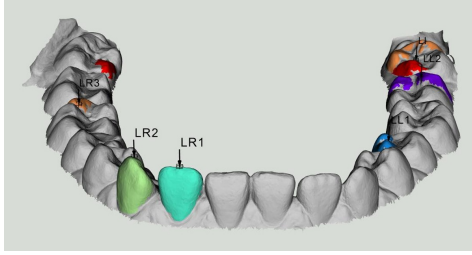


Figure 24: The same model as shown in figure 23 but with the bottom cropped off. Any tooth which now touched the edge of the scan is rejected in section 3.3.4.

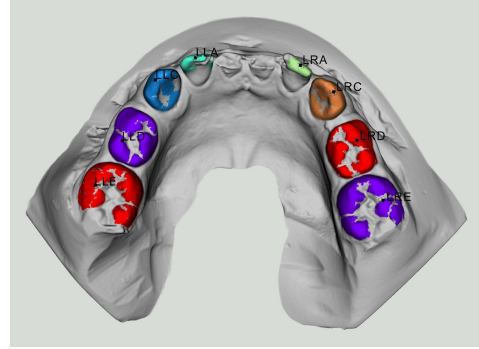


Figure 26: Neither the software, nor the person who programmed it, can tell lower incisors apart.

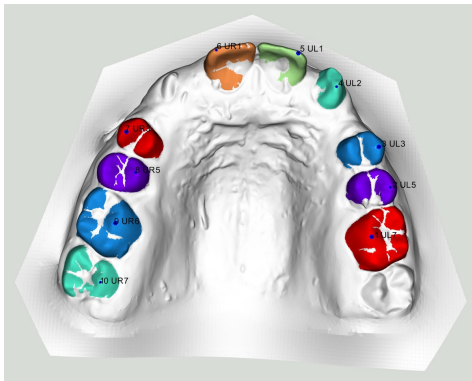


Figure 25: A model with several teeth misassigned. Section 3.5.3 needs some work.

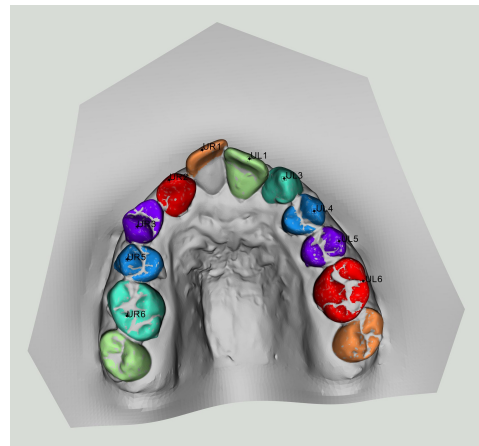


Figure 27: The orientation of the UR4 causes it to be mistaken for an UR3.

from central lower incisors, this is hardly surprising.

All but one of the tooth characteristics in section 3.5.3 are dependent on the orientation of the teeth. It's rare that a tooth be oriented unusually enough to cause a different assignment but it can happen as it did in figure 27. This dependence also rules out any chance of recognising teeth *lose*, i.e. not part of a full model, which would be of great value to forensics. Ideally, characteristics should only measure orientation-independent properties such as curvature or normalise orientation first using something like PCA.

With all teeth present, however, linear programming is again able to keep the final conclusion correct (see figure 28).

## 5. Discussion

Regarding tooth partitioning, one of the previous works showed an advantage in relation to ours. [7] used teeth outlines to achieve teeth segmentation. One of the complications that arised from our methodology, was that molar teeth were often recognised as two teeth instead of one entity. Whilst our software is later able to correct this issue after identifying each tooth, the segmentation procedure by [7] doesn't make this error to begin with. Our software doesn't truly partition teeth, often leaving gaps in grooves on molars or ignoring the lingual surface of incisors. For finding landmarks this is not an issue but for forensic dentistry, true partitioning is requirement to achieve accurate identification. True





2. Use the local maxima in the vertical direction to automatically provide an initial approximation of the landmarks;
3. Extract surface gradient and curvature information to identify the shape and boundaries of individual tooth – developing a 3D image segmentation technique specific for the purpose of tooth segmentation;
4. Order teeth through a best-fit quadratic jaw-line approximation;
5. Use a combination of machine learning and linear programming to recognize and label each tooth and its landmarks.

We also provide quite a few prior attempts that were tried but didn't work so as to prevent any future developers from making the same mistakes. We have successfully automated the MHB scoring system by using the methods studied in this paper. (The MHB software's details and scoring performance will be reported elsewhere in future.) Furthermore, this software has a much broader application. It can be expanded to automatically identify landmarks for a range of other scoring indices.

## References

- [1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [2] D. B. Forsyth and D. N. Davis. Assessment of an automated cephalometric analysis system. *Eur. J. Orthod.*, 18(1):471–478, 1996.
- [3] D. Gray and P. A. Mossey. Evaluation of a modified Huddart/Bodenham scoring system for assessment of maxillary arch constriction in unilateral cleft lip and palate subjects. *Eur. J. Orthod.*, 27(5):507–511, 2005.
- [4] B. ji Zou, S. jian Liu, S. hui Liao, X. Ding, and Y. Liang. Interactive tooth partition of dental mesh base on tooth-target harmonic field. *Computers in Biology and Medicine*, 56:132 – 144, 2015.
- [5] E. Kalogerakis, A. Hertzmann, and K. Singh. Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics*, 29(3), 2010.
- [6] V. A. Knyaz and A. V. Gaboutchian. Photogrammetry-based automated measurements for tooth shape and occlusion analysis. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B5:849–855, 2016.
- [7] T. Kronfeld, D. Brunner, and G. Brunnett. Snake-based segmentation of teeth from virtual dental casts. *Computer-Aided Design and Applications*, 7(2):221–233, 2010.
- [8] Y. Kumar, R. Janardan, and B. Larson. Automatic feature identification in dental meshes. *Computer-Aided Design and Applications*, 9:747–769, 08 2013.
- [9] Z. Li, X. Ning, and Z. Wang. A fast segmentation method for stl teeth model. In *2007 IEEE/ICME International Conference on Complex Medical Engineering*, pages 163–166, 2007.
- [10] X. Ma, C. Martin, G. McIntyre, P. Lin, and P. Mossey. Digital Three-Dimensional automation of the modified huddart and bodenham scoring system for patients with cleft lip and palate. *Cleft Palate Craniofac. J.*, 54(4):481–486, July 2017.
- [11] P. A. Mossey, J. D. Clark, and D. Gray. Preliminary investigation of a modified Huddart/Bodenhamx scoring system for assessment of maxillary arch constriction in unilateral cleft lip and palate subjects. *Eur. J. Orthod.*, 25(3):251–257, June 2003.
- [12] D. J. Rudolph, P. M. Sinclair, and J. M. Coggins. Automatic computerized radiographic identification of cephalometric landmarks. *Am. J. Orthod. Dentofacial Orthop.*, 113(2):173–179, Feb. 1998.
- [13] K. Wu, L. Chen, J. Li, and Y. Zhou. Tooth segmentation on dental meshes using morphologic skeleton. *Computers & Graphics*, 38:199 – 211, 2014.
- [14] Y. J. Chen, S. Kuang, C. Dds, H. Fu, C. Dds, and K. Chee. Comparison of landmark identification in traditional versus Computer-Aided digital cephalometry. *The Angle Orthodontist*, 70(5):387–392, 2000.
- [15] T. Yuan, W. Liao, N. Dai, X. Cheng, and Q. Yu. Single-tooth modeling for 3d dental model. *International journal of biomedical imaging*, 2010, 06 2010.

## Appendix A. Calculating PCA

Information describing how to calculate PCA is rather sparsely available. Hence, a recipe to apply PCA to a set of points is included below.

Consider all points in the model.

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$$

Subtract the centre of mass from each point to get displacements. The centre of mass being the mean of

all points.

$$X' = \{\mathbf{x}_i - \bar{\mathbf{X}}, i = 1 \dots n\}$$

Matrix multiply  $X'$  transposed with itself.

$$M = X'^T X'$$

$M$  should be a  $3 \times 3$  matrix and is typically referred to as the *covariance matrix*.

Then use eigen decomposition on  $M$  to get three eigenvalues and their corresponding eigenvectors. The eigenvectors are the unit-vectors / directions and the eigenvalues are the covariances in those directions. The unit-vectors should be sorted by eigenvalue from largest to smallest.